Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Trabalho Prático 3: simulando o sistema de arquivos FAT de 16 bits

Rodrigo José Zonzin Esteves

1 Introdução

A proposta do trabalho é simular o sistema de arquivos File Allocation Table de 16 bits. Nosso interesse é manipular apenas as estruturas que tratam arquivos e diretórios. Dessa forma, a seguinte simplificação foi adotada: não temos que nos preocupar com o boot block (mantendo-o sempre com o valor 0xbb). Também temos um número limitado de entradas de diretório: 32. Para tal projeto, temos as seguintes especificações:

1. Tamanho do setor: 512 bytes
2. Tamanho do cluste: 2 setores $= 2 \cdot 512$ bytes $= 1024$ bytes
3. Clustes: 4096 (ou seja, $4096 \cdot 1024$ bytes) $= 4194304$ bytes

O sistema FAT tem a seguinte característica: um boot block de 2 setores (1 cluster) com valores default Oxbb, uma seção FAT de 8 clusters com valor inicial Oxfffd, os 9 primeiros valores 0xfffe, o décimo valor com Oxffff e todos os demais inicializados em 0x0000. O diretório root é inicializado em zero para todas as suas posições e os clusters de dados também!

2 Estruturas de dados

As estruturas de dados necessárias para a implementação do trabalho já estavam disponibilizadas: entrada de diretório e cluster de dados.

A Entrada de Diretórios é um registro que contém informações pertinentes para a manipulação do arquivo: nome do arquivo, atributos, frimeiro bloco e tamanho. Essa estrutura de dados é a mais básica para a execução do trabalho, pois através dela podemos carregar para um *array* a região dos diretórios no arquivo *.part*, criar novos diretórios a serem escritos e então salvá-los no arquivo.

3 mkdir

Para criarmos um diretório dentro do diretório root, podemos carregar a seção de bytes equivalente a essa estrutura na FAT (*root_dir*), instanciar uma variável *dir_entry_t* com esses valores e atribuí-la à região em questão. De forma simplificada, fazemos algo da seguinte forma:

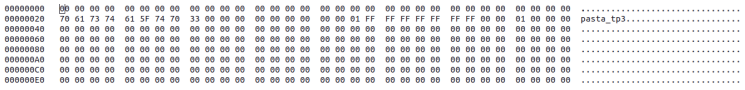
```
1 void create(){
2   FILE* ptr_file = fopen(...);
3   fseek(ptr_file, sizeof(boot_block), SEEK_SET);    //posiciona stream
4   fread(fat, sizeof(fat), 1, ptr_file);             //le o fat
5   fread(root_dir, sizeof(root_dir), 1, ptr_file);   //le o diretorio root
6
7   root_dir[0] = {"meu_diretorio", 12, {0, 12, 12,0,0,0,0}, 0x12, 12};
```

```
8     fwrite(&root_dir, sizeof(root_dir), 1, saida2);
9     fclose();
10 }
```

A figura a seguir mostra os estrados inicial do sistema FAT, inicializado em zero, e final, após a criação de um diretório "**pasta_tp3**".

Campo	Valor
Filename	"pasta_tp3"
Attributes	1
Reserved	{-1,-1,-1,-1,-1,-1,-1}
First Block	0x**** [1]
Size	1

(a) Antes



(b) Depois

Figura 1: Estado do root dir antes e depois da criação do diretório

Note que o atributo *first_block* deve apontar para uma posição livre do *data_cluster*, onde lá uma nova instância do tipo *dir_entry_t* pode ser criada para que um subdiretório seja criado.

4 ls

Inicialmente, implementamos a instrução *ls* para o diretório root. Para esse propósito, lemos para um array do tipo *dir_entry_t* todos os bytes contidos na partição relacionados à estrutura "diretório root".

Em seguida, percorremos todas as posições desse vetor printando o campo *"name"* da *struct*.

Na figura a seguir mostramos essa abordagem: criamos os seguintes diretórios: "pasta_3", "aluno", "usr", "bin"com atributos arbitrários. Em seguida, chamamos a função *ls*.

```
zonzin@rodrigo:~/Documentos/Faculdade/SO/tp3$ ./fat
Name: pasta_tp3
Size: 1
Attribute: 1

Name: rodrigo
Size: 2
Attribute: 1

Name: usr
Size: 1
Attribute: 1

Name: bin
Size: 200
Attribute: 1
```

Figura 2: Implementação de ls no diretório root

Note que todos os atributos foram setados como bit 1, pois todos são diretórios. Casou houvesse um arquivo, como “arquivo.txt”, o atributo relacionado teria esse bit em zero.

Para um ls partindo do diretório root e caminhando ”para dentro”de outros diretórios, devemos seguir a abordagem descrita em seguida.

1. Carrega o diretório root
2. Analisa as entradas de diretório contidas ali.
3. Percorre todas as 32 possíveis entradas, buscando aquela que contém o atributo “name”correspondente ao buscado.
4. Ao achá-lo, acesse o campo “primeiro_bloco”e busque o bloco apontado no cluster de data.
5. Carregando esse bloco, basta dar print no atributo “name”das entradas de diretório contidas naquela entrada de diretório.

Não conseguimos, no entanto, fazer uma implementação funcional desse comando, pois não conseguimos lidar com os ponteiros “primeiro_bloco”e as posições correspondentes na fat.

5 Create

De forma similar ao mkdir, para criarmos um arquivo, basta carregarmos os bytes correspondentes ao diretório root, à fat e ao data cluster para as variáveis globais.

Em seguida, criamos uma instância do tipo *dir_entry_t* onde podemos setar os dados relacionados ao arquivo: nome, atributo (0 para arquivo), tamanho e primeiro bloco.

Após esse procedimento, basta atribui-lo à posição desejada no cluster e realizar a persistência do arquivo.

```
00002760  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
00002778  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
00002790  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
000027A8  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
000027C0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
000027D8  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
000027F0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  61 72 71 75  69 76 6F 2E  ................arquivo.
00002808  74 78 74 00  00 00 00 00  00 00 00 FF  FF FF FF FF  FF FF 01 00  03 00 00 00  txt.....................
00002820  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
00002838  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
00002850  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
00002868  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
00002880  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ........................
```

Figura 3: Criação de um arquivo *txt*

Note que na figura 1(b) existe um byte ”0x01”1 posição antes dos bytes reservados (são definidos na struct mas nunca usados). Esse valor refere-se ao campo “atributo”e nos diz sobre a natureza do arquivo: um diretório.

Já na figura 3, temos um byte setado em zero antes da sequência de “0xFF”(-1 em decimal). Isso nos diz que o dado em questão é um arquivo simples, e não um diretório.

```
1   void create(){
2     FILE *ptr_file;
3
4     ptr_file = fopen(fat_name, "r+");               //abre o arquivo para leitura e escrita
5     fseek(ptr_file, sizeof(boot_block), SEEK_SET);  //posiciona stream apos o bootblock
6     fread(fat, sizeof(fat), 1, ptr_file);           //le o fat
7     fread(root_dir, sizeof(roo_dir), 1, prt_file);  //le o diretorio root
8     fread(clusters, sizeof(clusters), 1, ptr_file); //le os data clusters
9
10    dir_entry_t new_file = {"arquivo.txt", 0, {-1,-1,-1,-1,-1,-1,-1}, 0x001, 3};
11    clusters->dir[0] = new_file;
12
13    fwrite(&fat, sizeof(fat), 1, ptr_file);           //salva no disco
14    fwrite(&root_dir, sizeof(root_dir), 1, ptr_file); //salva no disco
```

```
15      fwrite(&clusters, sizeof(clusters), 1, ptr_file);  //salva no disco
16
17      fclose(ptr_file);
18    }
```

6 O que não foi implementado

As funções unlink, write, append e read não foram implementadas. Mesmo aquelas que foram implementadas apresentam sérias limitações:

1. A função create apenas instancia o nome do arquivo e seus atributos em uma região do sistema de arquivos. Não há link para onde os dados do arquivo seriam armazenados.
2. A função ls (chamada de *ls_root* no código) só é capaz de listar os arquivos contidos no diretório raiz. Qualquer subdiretório dentro do diretório raiz não é passível de acesso (pois, mais uma vez, não conseguimos linkar o conteúdo do arquivo com uma região válida para armazenamento)
3. A função mkdir só consegue criar diretórios no próprio diretório raiz.

7 Conclusão

O código é pouco funcional e está longe de atender às demandas do trabalho. Sua implementação teve, no fim, um papel estritamente elucidativo sobre as nuances do sistema FAT16. A despeito da tentativa, a qualidade do trabalho (tanto o código quanto a documentação) não está em linha com a expectativa de uma simulação fidedigna do sistema de arquivos do clássico MS-DOS. Mas, citando Tanenbaum (2016, p. 200), "a vida é assim".

Referências

Tanenbaum, A. S. (2016). *Sistemas Operacionais modernos*. Pearson, 4 th edition.